Webmaster in a Nutshell, 2nd Edition
By Robert Eckstein, Stephen Spainhour

Table of Contents

Part III: XML

Chapter 10. XML

The Extensible Markup Language (XML) is a document processing standard proposed by the World Wide Web Consortium (W3C), the same group responsible for overseeing the HTML standard. Although the exact specifications have not been completed yet, many expect XML and its sibling technologies will replace HTML as the markup language of choice for dynamically generated content, including nonstatic web pages. Already several browser and word processor companies are integrating XML support into their products.

XML is actually a simplified form of Standard Generalized Markup Language (SGML), an international documentation standard that has existed since the 1980s. However, SGML is extremely bulky, especially for the Web. Much of the credit for XML's creation can be attributed to Jon Bosak of Sun Microsystems, Inc., who started the W3C working group responsbile for scaling down SGML to a form more suitable for the Internet.

Put succinctly, XML is a *meta-language* that allows you to create and format your own document markups. With HTML, existing markup is static: `<HEAD>` and `<BODY>`, for example, are tightly integrated into the HTML standard and cannot be changed or extended. XML, on the other hand, allows you to create your own markup tags and configure each to your liking: for example, `<HeadingA>`, `<Sidebar>`, or `<Quote>`, or even `<ReallyWildFont>`. Each of these elements can be defined through your own *document type definitions* and *stylesheets* and applied to one or more XML documents. Thus, it is important to realize that there are no "correct" tags for an XML document, except those you define yourself.

While many XML applications currently support CSS, a more extensible stylesheet specification exists called the *Extensible Stylesheet Language* (XSL). By using XSL, you ensure that your XML documents are formatted the same no matter which application or platform they appear on. This chapter documents portions of the XSL draft standard dated 16 December 1998, which marks a significant revision over its predecessors.

This chapter offers a quick overview of XML, as well as some sample applications that allow you to get started in coding. We won't cover everything about XML. In fact, much of XML is still in flux as this book goes to print. Consequently, creating a definitive reference at this point in XML's life seems a bit futile. However, after reading this chapter, we hope that the components that make up XML will seem a

Hardware
Human-Computer Interaction
Internet/Online
IT Management
Markup Languages
Multimedia
Networking
Operating Systems
Programming
Security
Software Engineering

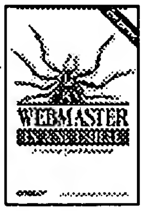
Find a Specific Book

- Author
- ISBN
- Title
- Publisher

little less foreign to you.

Print E-Mail Add Note Add Bookmark Previous Next Top

URL http://proquest.safaribooksonline.com/1565923251/wn.ch10.div.1



Webmaster in a Nutshell, 2nd Edition
By Robert Eckstein, Stephen Spainhour

Table of Contents

Chapter 10. XML

10.1. XML Terminology

Before we move further, we need to standardize some terminology. An XML document consists of one or more *elements*. An element is marked with the following form:

```
<Body>This is text formatted according the the Body element</Body>
```

This element consists of two *tags*, an opening tag which places the name of the element between a less-than (<) and a greater-than (>) sign, and a closing tag which is identical except for the forward slash (/) that appears before the element name. Like HTML, the text contained between the opening and closing tags is considered part of the element and is formatted according to the element's rules.

Elements can have *attributes* applied to them, such as the following:

```
<Price currency="Euro" >25.43</Price>
```

Here, the attribute is specified inside of the opening tag and is called "currency." It is given a value of "Euro", which is expressed inside quotation marks. Attributes are often used to further refine or modify the default behavior of an element.

In addition to the standard elements, XML also supports *empty elements*. An empty element has no text appearing between the opening and closing tag. Hence, both tags can (optionally) be merged together, with a forward slash appearing before the closing marker. For example, the following elements are identical:

```
<Picture src="blueball.gif" ></Picture>
<Picture src="blueball.gif"/>
```

Empty elements are often used to add nontextual content to a document, or to provide additional information to the application that is parsing the XML. Note that while the closing slash may not be used in single-tag HTML elements, it is *mandatory* for single-term XML empty elements.

10.1.1 Unlearning Bad Habits

Hardware
Human-Computer Interaction
Internet/Online
IT Management
Markup Languages
Multimedia
Networking
Operating Systems
Programming
Security
Software Engineering

Find a Specific Book

- Author
- ISBN
- Title
- Publisher

Whereas HTML browsers often ignore simple errors in documents, XML applications are not nearly as forgiving. For the HTML reader, there are a few bad habits that we should first disuade you of:

Attribute values must be in quotation marks.

You can't specify an attribute value such as `<picture src=/images/blueball.gif>`, which HTML browsers often overlooked. An attribute value must always be inside single or double quotation marks, or the XML parser will flag it as an error. Here is the correct way to specify such a tag:

```
<picture src="/images/blueball.gif"
>
```

An element must either have an opening and closing tag, or be an empty element.

Each element that specifies an opening tag must have a closing tag that matches it. If it does not, and it is not an empty element, the XML parser generates an error. In other words, you cannot do the following:

```
<Paragraph>
This is a paragraph.
<Paragraph>
This is another paragraph.
```

Instead, you must have an opening and closing tag for each paragraph element:

```
<Paragraph>This is a paragraph.</Paragraph>
<Paragraph>This is another paragraph.</Paragraph>
```

Tags must be nested correctly.

It is illegal to do the following:

```
<Italic><Bold>This is incorrect</Italic></Bold>
```

The closing tag for the `Bold` element should be inside the closing tag for the `Italic` element, to match the nearest opening tag and preserve the correct element nesting. It is essential for the application parsing your XML to process the hierarchy of the elements:

```
<Italic><Bold>This is correct</Bold></Italic>
```

These syntactic rules are the source of many common errors in XML, especially given that some of this behavior can be ignored by HTML browsers. An XML document that adheres to these rules (and a few others which we'll see later) is said to be *well-formed*.

10.1.2 An Overview of an XML Document

There are generally three files that are processed by an XML-compliant application to display XML content. They are:

The XML document

This file contains the document data, typically tagged with meaninful XML elements, some of which may contain attributes.

A stylesheet

The stylesheet dictates how document elements should be formatted when they are displayed, whether it be in a word processor or a browser. Note that you can apply different stylesheets to the same document, depending on the environment, thus changing its appearance without affecting any of the underlying data. The separation between content and formatting is an important distinction in XML.

Document Type Definition (DTD)

This file specifies rules for how the XML document elements, attributes, and other data are defined and logically related in an XML-compliant document.

10.1.3 A Simple XML Document

Example 10.1 shows a simple XML document.

Example 10.1. simple.xml

<?xml version="1.0" standalone="no"?> <!DOCTYPE OReilly:Books SYSTEM "sample.dtd"> <!-- Here begins the XML data --> <OReilly:Books xmlns:OReilly='http://www.oreilly.com/'> <OReilly:Product>Webmaster in a Nutshell</OReilly:Product> <OReilly:Price>19.99</OReilly:Price> </OReilly:Books>

Let's look at this example line by line.

In the first line, the code between the `<?xml` and the `?>` is called an *XML declaration*. This declaration contains special information for the XML processor (the program reading the XML) indicating that this document conforms to Version 1.0 of the XML standard. In addition, the `standalone="no"` attribute informs the program that an outside DTD is needed to correctly interpret the document. (In this case, the DTD will reside in a separate file called *sample.dtd*.) On a side note, it is possible to simply embed the stylesheet and the DTD in the same file as the XML document. However, this is not recommended for general use, as it hampers reuse of both DTDs and stylesheets.

The second line is as follows:

```
<!DOCTYPE OReilly:Books SYSTEM "sample.dtd">
```

This line points out the *root element* of the document, as well as the DTD that validates each of the document elements that appear inside the root element. The root element is the outermost element in the document that the DTD applies to; it typically denotes the document's starting and ending point. In this example, the `<OReilly:Books>` element serves as the root element of the document. The `SYSTEM` keyword denotes that the DTD of the document resides in a separate local file named `sample.dtd`.

Following that line is a comment. Comments always begin with `<!--` and end with `-->`. You can write whatever you want inside a comment; they are ignored by the XML processor. Be aware that comments, however, cannot come before the XML declaration and cannot appear inside of an element tag. For example, this is illegal:

```
<OReilly:Books <!-- This is the tag for a book --!>>
```

Finally, `<OReilly:Product>`, `<OReilly:Price>`, and `<OReilly:Book>` are XML elements we invented. Like most elements in XML, they hold no special significance except for whatever document and style rules we define for them. Note that these elements look slightly different than those you may have seen previously because we are using *namespaces*. Each element tag can be divided into two parts. The portion before the colon (:) forms the tag's namespace, while the portion after the colon identifies the name of the tag itself.

Some XML terminology here: the `<OReilly:Product>` and `<OReilly:Price>` elements would consider the `<OReilly:Books>` element their *parent*. In the same manner, elements can be grandparents and grandchildren of other elements. However, we typically abbreviate multiple levels by stating that an element is either an *ancestor* or a *descendant* of another element.

10.1.3.1 Namespaces

Namespaces are a recent addition to the XML specification. The use of namespaces is not mandatory in XML, but it's often wise. Namespaces were created to ensure uniqueness among XML elements.

For example, let's pretend that the `<OReilly:Books>` element was simply named `<Books>`. When you think about it, it's not out of the question that another publisher would create its own `<Books>` element in its own XML documents. If the two publishers combined their documents, resolving a single (correct) definition for the `<Books>` tag would be impossible. When two XML documents containing identical elements from different sources are merged, those elements are said to *collide*. Namespaces help to avoid element collisions by scoping each tag.

In Example 10.1, we scoped each tag with the `OReilly` namespace. Namespaces are declared using the `xmlns:`*something* attribute, where *something* defines the ID of the namespace. The attribute value is a unique identifier that differenciates it from all other namespaces; the use of a URI is recommended. In this case, we use the O'Reilly URI `http://www.oreilly.com/` as the default namespace, which should guarantee uniqueness. A namespace declaration can appear as an attribute of any element, so long as the namespace's use remains inside that element's opening and closing tags. Here are some examples:

```
<OReilly:Books xmlns:OReilly='http://www.oreilly.com/'>
<xsl:stylesheet xmlns:xsl='http://www.w3.org/'>
```

You are allowed to define more than one namespace in the context of an element:

```
<OReilly:Books xmlns:OReilly='http://www.oreilly.com/'
               xmlns:Songline='http://www.songline.com/'>
...
</OReilly:Books>
```

If you do not specify a name after the `xmlns` prefix, the namespace is dubbed the *default namespace* and is applied to all elements inside the defining element that do not use a namespace prefix of their own. For example:

```
<Books xmlns='http://www.oreilly.com/'
       xmlns:Songline='http://www.songline.com/'>
    <Book>
        <Title>Webmaster in a Nutshell</Title>
        <ISBN>1-56592-229-8</ISBN>
    </Book>
    <Songline:CD>18231</Songline:CD>
</Books>
```

Here, the default namespace (represented by the URI `http://www.oreilly.com/`) is applied to the elements `<Books>`, `<Book>`, `<Title>`, and `<ISBN>`. However, it is not applied to the `<Songline:CD>` element, which has its own namespace.

Finally, you can set the default namespace to an empty string to ensure that there is no default namespace in use within a specific element:

```
<header xmlns=''
        xmlns:OReilly='http://www.oreilly.com/'
        xmlns:Songline='http://www.songline.com/'>
    <entry>Learn XML in a Week</entry>
    <price>10.00</price>
</header>
```

Here, the `<entry>` and `<price>` elements have no default namespace.

10.1.4 A Simple Document Type Definition (DTD)

Example 10.2 creates a simple DTD for our XML document.

Example 10.2. simple.dtd

<!-- DTD for sample document --> <!ELEMENT OReilly:Books (OReilly:Product, OReilly:Price)> <!ELEMENT OReilly:Product (#PCDATA)> <!ELEMENT OReilly:Price (#PCDATA)>

The purpose of the this DTD is to declare each of the elements used in our XML document. All document-type data is placed inside a construct with the characters `<!`*something*>. Like the previous XML example, the first line is a comment because it begins with `<!--` and ends with the characters `-->`.

The `<!ELEMENT>` construct declares each valid element for our XML document. With the second line, we've specified that the `OReilly:Books` element is valid:

```
<!ELEMENT OReilly:Books (OReilly:Product, OReilly:Price)>
```

The parentheses group required child elements for the `<OReilly:Books>` element. In this case, the `<OReilly:Product>` element and the `<OReilly:Price>` element *must* be included inside our `<OReilly:Books>` element tags, and they must appear in the order specified. `<OReilly:Product>` and `<OReilly:Price>` are *children* of `<OReilly:Books>`.

Likewise, both the `<OReilly:Product>` and the `<OReilly:Price>` elements are declared in our DTD:

```
<!ELEMENT OReilly:Product (#PCDATA)>
<!ELEMENT OReilly:Price (#PCDATA)>
```

Again, parentheses specify required elements. In this case, they both have a single requirement, which is represented by `#PCDATA`. This is shorthand for *parsed character data*, which means that any characters are allowed, so long as they do not include other element tags or contain the characters `<` or `&`, or the sequence `]]>`. These characters are forbidden because they could be interpreted as markup. (We'll see how to get around this shortly.)

The XML data shown in Example 10.2 adheres to the rules of this DTD: it contains an `<OReilly:Books>` element, which in turn contains an `<OReilly:Product>` element, followed by an `<OReilly:Price>` element inside it (in that order). Therefore, if this DTD is applied to it with a `<!DOCTYPE>` statement, the document is said to be *valid.*

So far, we've structured the data but haven't paid much attention to its formatting. Now let's move on and add some style to our XML document.

10.1.5 A Simple XSL Stylesheet

The Extensible Stylesheet Language consists of a series of markups that can be used to apply formatting rules to each of the elements inside an XML document. XSL works by applying various style rules to the contents of an XML document, based on the elements that it encounters.

Let's add a simple XSL stylesheet to the example:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
                xmlns:fo="http://www.w3.org/TR/WD-xsl/FO">
  <xsl:template match="/">
       <fo:block font-size="18pt">
         <xsl:apply-templates/>
       </fo:block>
  </xsl:template>
</xsl:stylesheet>
```

The first thing you might notice when you look at an XSL stylesheet is that it is formatted in the same way as a regular XML document. This is not a coincidence. In fact, by design XSL stylesheets are themselves XML documents, so they must adhere to the same rules as well-formed XML documents.

Breaking down the pieces, you should first note that all XSL elements must be enclosed in appropriate `<xsl:stylesheet>` tags. These tags tell the XSL processor that it is describing stylesheet information, not XML content itself. Between the `<xsl:stylesheet>` tags lie each of the rules that will be applied to our XML document. Each of these rules can be further broken down into two items: a *template pattern* and a *template action*.

Consider the line:

```
<xsl:template match="/" >
```

This line forms the template pattern of the stylesheet rule. Here, the target pattern is the root element, as designated by `match="/"`. The `"/"` is shorthand to represent the XML document's root element (`<OReilly:Books>` in our case). Remember that if this stylesheet is applied to another XML document, the root element matched might be different.

The following lines:

```
<fo:block font-size="18pt" >
  <xsl:apply-templates/>
</fo:block>
```

specify the template action that should be performed on the target. In this case, we see the empty element `<xsl:apply-templates/>` located inside a `<fo:block>` element. When the XSL processor formats the target element, every element that is inside the root element's opening and closing tags uses an 18-point font.

In our example, the `<OReilly:Product>` and `<OReilly:Price>` elements are enclosed inside the `<OReilly:Books>` tags. Therefore, the font size will be applied to the contents of those tags.

Example 10.3 displays a more realistic example.

Example 10.3. simple.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"
                xmlns:fo="http://www.w3.org/TR/WD-xsl/FO">
                xmlns:OReilly="http://www.oreilly.com/">
  <xsl:template match="/">
      <fo:display-sequence>
        <xsl:apply-templates/>
      </fo:display-sequence>
  </xsl:template>
  <xsl:template match="OReilly:Books">
      <fo:block font-size="18pt">
        <xsl:text>Books:</xsl:text>
        <xsl:apply-templates/>
      </fo:block>
  </xsl:template>
  <xsl:template match="OReilly:Product">
      <fo:block font-size="12pt">
        <xsl:apply-templates/>
      </fo:block>
  </xsl:template>
  <xsl:template match="OReilly:Price">
      <fo:block font-size="14pt">
```

```
            <xsl:text>Price: $</xsl:text>
            <xsl:apply-templates/>
            <xsl:text> + tax</xsl:text>
        </fo:block>
    </xsl:template>
</xsl:stylesheet>
```

In this example, we're now targeting the `<OReilly:Books>` element, printing the word "Books:" before it in an 18-point font. In addition, the `<OReilly:Product>` element now applies a 12-point font to each of its children, and the `<OReilly:Price>` tag now uses a 14-point font to display its children, overriding the default 18-point font of its parent, `<OReilly:Books>`. (Of course, neither one has any children elements; they simply have text between their tags in the XML document.) The text "`Price: $`" will now precede each of `<OReilly:Price>`'s children, and the characters " `+ tax`" will now come after it, formatted accordingly. [3]

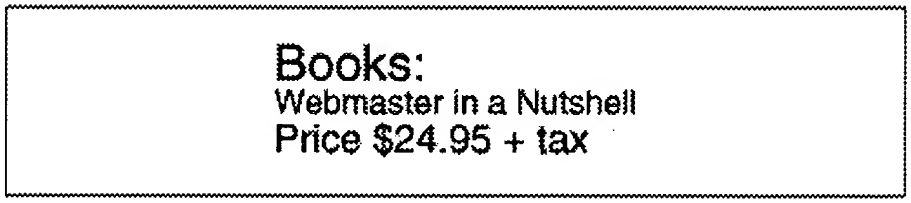
[3] You may have noticed that we are using the `<fo:display-sequence>` element instead of `<fo:block>` for the root element. This is primarily because the pattern matching our root element really doesn't do anything anymore. However, you needn't be concerned with this here.

Here is the result after we pass it through an XSL processor:

```
<?xml version="1.0"?>
<fo:display-sequence>
  <fo:block font-size="18pt">
Books:
    <fo:block font-size="12pt">
Webmaster in a Nutshell
    </fo:block>
    <fo:block font-size="14pt">
Price $19.99 + tax
    </fo:block>
  </fo:block>
</fo:display-sequence>
```

And that's it: everything needed for a simple XML document! Running it through an XML processor, you should see something similar to Figure 10.1.

Figure 10.1. Sample XML output


Books:
Webmaster in a Nutshell
Price $24.95 + tax